Insertion Sort

Introduction

Insertion sort iterates, consuming one input element each repetition, and grows a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain [4].

Performance

In general, the performance of the sorting algorithm is analyzed from two aspects, time complexity and space complexity. Time complexity represents the time taken to execute the algorithm, which is generally considered in three cases: best case, worst case, and average case [6]. Space complexity represents the amount of memory required to complete a program [5].

Use **array** as data structure and **n** denotes the input array size, insertion sort performance is as follows [4]:

Worst-case time complexity	$\boldsymbol{O(n^2)}$
Average time complexity	$\boldsymbol{O(n^2)}$
Best-case time complexity	$\boldsymbol{O(n)}$
Worst-case space complexity	$\boldsymbol{O(n)}$

Pseudocode [3]

Algorithm: InsertionSort(Arr)

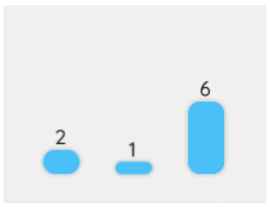
Input: an array of integers Arr

Output: The result of sorting Arr

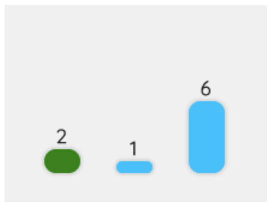
```
length = Arr.length
  for i = 1 to length -1 do
    preIndex = i - 1
    current = arr[i]
    while preIndex >= 0 && current < Arr[preIndex] do
        Arr[preIndex + 1] = Arr[preIndex]
        preIndex--
    end while
  Arr[preIndex + 1] = current
  end for
  return Arr
```

Example

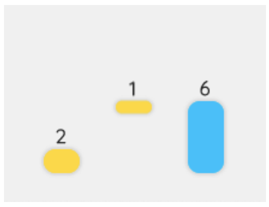
Sorting the list: 2,1,6



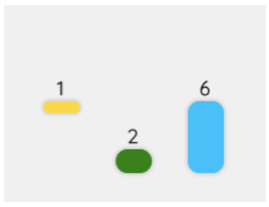
(1)



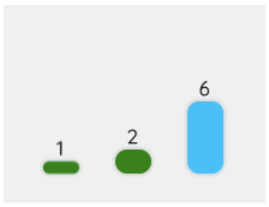
(2)



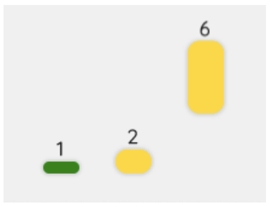
(3)



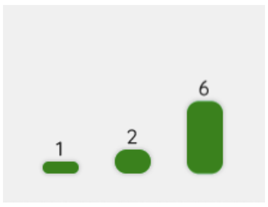
(4)



(6)



(7)



(8)

Implement in programming language

Java [1]

```java
public void insertionSort(int arr[])
    {
        int n = arr.length;
        for (int i=1; i<n; ++i)
        {
            int key = arr[i];
            int j = i-1;

            while (j>=0 && arr[j] > key)
            {
                arr[j+1] = arr[j];
                j = j-1;
            }
            arr[j+1] = key;
        }
    }
```

JavaScript [2]

```
function insertionSort(arr) {
    var len = arr.length;
    var preIndex, current;
    for (var i = 1; i < len; i++) {
        preIndex = i - 1;
        current = arr[i];
        while(preIndex >= 0 && arr[preIndex] > current) {
            arr[preIndex+1] = arr[preIndex];
            preIndex--;
        }
        arr[preIndex+1] = current;
    }
    return arr;
}
```

C [2]

```
void insertion_sort(int arr[], int len){
        int i,j,key;
        for (i=1;i<len;i++){
                key = arr[i];
                j=i-1;
                while((j>=0) && (arr[j]>key)) {
                        arr[j+1] = arr[j];
                        j--;
                }
                arr[j+1] = key;
        }
}
```

References

1. GeeksforGeeks Contributors (2018). *Java Program for Insertion Sort.* [Online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/java-program-for-insertion-sort/ [Accessed 6 Mar. 2021].
2. Runoob Contributors (2018). *Insertion sort*. [Online] Runoob. Available at: https://www.runoob.com/w3cnote/insertion-sort.html [Accessed 6 Mar. 2021].
3. Visualgo Contributors (2014). [Software] Visualgo. Available at: https://visualgo.net/en/sorting [Accessed 6 Mar. 2021].
4. Wikipedia Contributors (2021). *Insertion sort*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Insertion_sort [Accessed 6 Mar. 2021].
5. Wikipedia Contributors (2021). *Space complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Space_complexity [Accessed 20 Mar. 2021].
6. Wikipedia Contributors (2021). *Time complexity*. [Online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Time_complexity [Accessed 20 Mar. 2021].